图、遍历

2019年2月24日 14:07

-
- 图的定义

1. 图G（Graph）由两个集合V和E组成，记为G=(V,E)
   1) 其中V是顶点（Vertex）的非空有穷集合，E是用顶点对表示的边（Edge）的有穷集合
   2) 在图结构中，每个结点既可以有多个直接前驱，也可以有多个直接后继。前面讨论的线性表和树都可以看成是两种特殊的图
   3) 无向图：表示边的顶点对是无序的
      i. 通常用（vi，vj）表示顶点vi和vj间相连的边
      ii. 在有n个顶点的无向图中，e的范围为0 - n(n-1)/2。 具有n(n-1)/2条边的无向图称为无向完全图
   4) 有向图：表示边的顶点对是有序的
      i. 有向边通常称为弧(Arc)，用<vi，vj>表示从顶点vi到顶点vj 的一条弧
      ii. 起点vi为**弧尾/初始点**，终点vj为**弧头/终端点**
      iii. 在有n个顶点的有向图中，e的范围为0 - n(n-1)，具 有n(n-1)条边的有向图称为有向完全图
2. 权（weight）：图的边或弧附有的相关数值
   1) 用于表示从一个顶点到另一个顶点的距离、时间耗费、开销等
   2) 网络（network）：每条边或弧都带权的图
3. 度：图的边或弧的数目等于顶点度数之和的一半
   1) 无向图中，顶点的度是依附于该顶点的边数
   2) 有向图中，入度是以该顶点为弧头的弧数，出度是以该顶点为弧尾的弧数
4. 子图：若G1={V1,E1},G2={V2,E2}是两个图，且V2被包含在V1中，E2被包含在E1中，则称图G2是图G1的子图
5. 路径：首尾相接并且无重复边 /弧的边/弧序列
   1) 路径长度：序列中的边/弧数
   2) 简单路径：除起点和终点以外，所有顶点彼此各不相同的路径
   3) 回路：起点和终点都是同一顶点的路径
   4) 简单回路：由简单路径组成的回路
6. 连通：若任意二个顶点之间均存在路径，则称图G为连通图
   1) 连通分量：在非连通的无向图G中，极大连通子图称为无向图的连通分量
   2) 强连通：在有向图G中，若顶点vi到顶点vj有路径存在，并且从顶点vj到顶点vi也有路径存在，则称vi到vj是强连通的
   3) 强连通图：在有向图G中，若任意二个顶点之间都是强连通的，则称G为~
   4) 强连通分量：在非强连通的有向图G中的极大强连通子图

-
- 图的存储结构

1. 邻接矩阵表示法（数组法）
   1) 邻接矩阵（Adjacency Matrix）是表示顶点之间相邻关系的矩阵
   2) 设G=（V，E）是具有n个顶点的图，则G的邻接 矩阵有如下性质的n阶矩阵：

$$A[i,j]=\begin{cases}1: & 若(v_i,v_j)\ 或\langle v_i,v_j\rangle 是E(G)中的边\\ 0: & 若(v_i,v_j)\ 或\langle v_i,v_j\rangle 不是E(G)中的边\end{cases}$$

      i.

$$A[i,j]=\begin{cases}W_{ij}; & 若(v_i,v_j)\ 或\langle v_i,v_j\rangle \in E(G)\\ 0或\infty; & 若(v_i,v_j)\ 或\langle v_i,v_j\rangle \notin E(G)\end{cases}$$

   3) 无向图的邻接矩阵一定是对称的，而有向图的不一定
   4) 无向图邻接矩阵的第i行(或第i列)非零元素个数正好是第i个顶点的度TD(vi)；有向图，邻接矩阵第i行非零元素个数正好是第i个顶点的出度OD(vi)，第i列非零元素的个数正好是第i个顶点的入度ID(vi)
   5) 邻接矩阵的行表示顶点的出边，邻接矩阵的列表示顶点的入边，因此从邻接矩阵很容易确定图中任意两个顶点间是否有边（或弧）相连
2. 建立无向图的邻接矩阵
   1)
```
#define VEX_NUM 5 /*顶点数目*/
#define ARC_NUM 6 /*图中边或弧数*/
typedef char Vextype; /*顶点类型*/
typedef struct {
        Vextype vexs[VEX_NUM]; /*顶点向量*/
        int arcs[VEX_NUM][VEX_NUM];
}Mgraph;/*邻接矩阵*/
```
   2)
```
void creat_Mgraph(Mgraph *G) { /*建立无向图的邻接矩阵G */
        for (i = 0; i < VEX_NUM; ++i)
                scanf("%c", &G->vexs[i]); /*输入顶点信息*/
        for (i = 0; i < VEX_NUM; ++i) /*初始化*/
                for (j = 0; j < VEX_NUM; ++j)
                        G->arcs[i][j] = 0;
        for (k = 1; k <= ARC_NUM; k++) {
                scanf("%d,%d", &i, &j);  /*输入表示边（vi,vj）的顶点序号i，j*/
                G->arcs[i][j] = 1;
                G->arcs[j][i] = 1; } } /* creat_Mgraph */
```
3. 邻接表(Adjacecy List) 是一种顺序分配和链式分配相结合的存储结构
   1) 每个顶点建立一个单链表，第i个单链表中的结点表示依附于顶点vi的所有的边（对有向图是以顶点vi为弧尾的弧）
   2) 每个链表上附设一个表头结点。在表头结点中，设有指向表中第一个结点的链域（firstarc）和存储顶点vi的名或者其他有关信息的数据域（data）
4. 建立有向图的邻接表
   1)
```
void creat_ALgraph( ALgraph G) { /* 建立有向图的邻接表G */
        for(i=0;i<VEX_NUM;++i) {
                scanf("%c",&G[i].data);   /*输入顶点信息*/
                G[i].firstarc=NULL; }
        for(k=1;k<=ARC_NUM;k++) {
```

```
scanf("%d,%d",&i,&j); /*输入表示弧,<vi,vj>的顶点序号i,j*/
p=(ArcNode *)malloc(sizeof(ArcNode));
p->adjvex=j;
p->nextarc=G[i].firstarc;
G[i].firstarc=p; } } /*creat_ALgraph */
```

2) 算法的时间复杂度为O（n+e）

-
- 图的遍历

1. 深度优先搜索：以图中某个顶点Vi为出发点，首先访问出发点，然后选择一个Vj，以Vj为新的出发点继续进行深度优先搜索，直至图中所有顶点都被访问过
   1) int visited[VEX_NUM] = {0};

```
void Dfs_m( Mgraph *G , int i) {
// 从第i个顶点出发深度优先遍历图G，G以邻接矩阵表示
    printf("%3c",G -> vexs[i]); /*访问顶点vi*/
    visited[i]=1;
    for (j=0; j<VEX_NUM; j++)
        if((G->arcs[i][j]==1)&& (!visited[j]))
            Dfs_m(G , j); } /*Dfs_m */
```

   2) int visited[VEX_NUM]={0};

```
void Dfs_L(ALgraph G,int i) {
// 从第i个顶点出发深度优先遍历图G，G以邻接表表示
    printf("%3c",G[i].data); /*访问顶点vi*/
    visited[i]=1;
    p=G[i].firstarc;
    while (p!=NULL) {
        if(visited[p->adjvex]==0)
            Dfs_L(G,p->adjvex);
        p=p->nextarc; } } /*dfs_L*/
```

2. 广度优先搜索：从图中某个顶点v出发，在访问了v之后，依次访问v的各个未曾访问过的邻接点；然后分别从这些邻接点出发，依次访问它们的未曾访问过的邻接点，直至所有顶点都被访问过
   1) int visited[VEX_NUM] = {0};

```
void Bfs(Mgraph G, int k) { // 从第k个顶点出发广度优先遍历
    InitQueue(Q);  // 访问顶点Vk
    printf("%3c" , G.vexs[k]);
    visited[k] = 1;
    EnQueue(Q,k);    // Vk入队列
    while(! QueueEmpty(Q) ) { //  队列非空
        DelQueue(Q , i);    //  队头顶点Vi出队列
        for( j=0;j<VEX_NUM; j++)
            if(G->arcs[i][j] ==1&&(!visited[j]) ) {
                printf("%3c" , G.vexs[j] ); // 访问顶点Vi未曾访问的顶点
                visited[j]=1;
                EnQueue(Q,j); /* Vj入队列*/} } }
```

      i.
      ii.
      iii.

iv.
v.
vi.
vii.
viii. ---------我是底线----------